20 | 推荐引擎：没有搜索词，“头条”怎么找到你感兴趣的文章？

2020-05-15 陈东

讲述：陈东
时长 22:34 大小 20.67M

你好，我是陈东。今天我来和你讲讲推荐引擎。

我们每天都会接触推荐引擎，最常见的，就是当我们用手机浏览资讯类 App 的时候，经常会用到的“下拉刷新”功能。你会发现，每次刷新之后，这些 App 都能给你推荐你最关心的“头条信息”。

那这些资讯类的 App，是怎么在没有搜索词的情况下，仅凭下拉刷新就可以在海量的文章中检索出你感兴趣的内容，并且推荐给你的呢？这就和推荐引擎中的检索技术有关了。今天，我就以资讯类 App 推荐文章为例，来和你聊一聊推荐引擎中的检索技术。

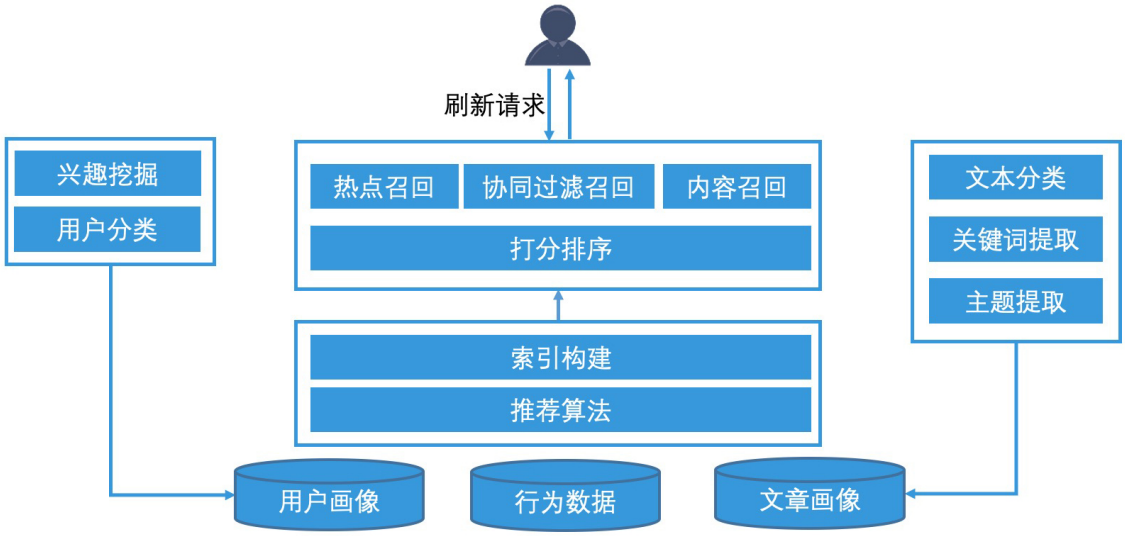
推荐引擎的整体架构和工作过程

我们知道，检索引擎的灵活程度和系统的检索约束条件有关。那我们先来看一下针对不同的引擎，系统的检索约束条件分别是什么。

在搜索引擎中，系统的强约束条件是用户输入的搜索词。而在广告引擎中，系统的强约束条件是广告主设置的定向要求。但是在资讯类 App 推荐引擎中，因为所有的用户操作只有“下拉刷新”这一个动作，所以外界输入的检索约束条件其实非常少。

因此，**相比于搜索引擎和广告引擎，推荐引擎具有更灵活的检索能力，也就是可以使用更灵活的检索技术，来进行文章的召回服务**。这也是推荐引擎相比于搜索引擎和广告引擎最大的不同之处。

那一个推荐引擎是怎么工作的呢？我按照功能划分，梳理出了推荐引擎的核心模块。



推荐引擎架构示意图

那接下来，我就结合这个架构图，来说说推荐引擎的核心工作流程。

首先，因为没有搜索词，所以推荐引擎并不能直接得知用户的意图和喜好。为了解决这个问题，推荐引擎会收集用户对不同文章的行为数据，包括曝光、点击、阅读、收藏、点赞和评论等等。

然后，我们会通过这些收集来的数据，在离线环节挖掘每一个用户兴趣，从而能对用户分类来生成完整的用户画像。在用户画像中，一个用户会拥有不同的标签，这些标签会有不同的

权重，所有的权重都会随着时间的变化而衰减。比如说，如果一个用户长时间没有继续这个行为，那标签就会逐步弱化。再比如说一个用户的兴趣发生变化了，那最新的兴趣标签的权重就会大于老的兴趣标签。通过这样的机制，我们就能更好地理解用户的喜好了。

但是，只给用户打上标签还不够，我们也要给文章打上标签。在这个过程中，我们除了要提取文章中的关键词以外，更多的要对文章中的内容做语义分析工作，比如，文章分类、主题词提取、主题提取等等。通过这些方式，推荐引擎就能为每一篇文章都生成文章画像了。

有了用户画像、文章画像，以及用户对文章的行为记录以后，我们就可以根据需求，灵活地使用不同的推荐算法来为用户推荐文章了。主要的推荐算法有 2 大类，分别是基于统计的静态召回算法和个性化召回算法。

所谓基于统计的静态召回，指的是根据当前系统对于文章的统计数据来进行推荐。比如说，我们可以在离线环节，提前统计好点击量最大、评论最多、收藏最多、收藏率上升最快的文章等。然后在线上环节将这些热门文章推荐给所有用户。它比较适合作为个性化召回不足时候的补充方案。

接下来，我们重点来讲讲个性化召回算法，因为一般来说，我们提到推荐算法的时候，指的都是个性化召回算法。个性化召回也有许多不同的方案，最有代表性的两种个性化召回，就是基于内容的召回（Content Based），以及基于协同过滤的召回（Collaborative Filtering）。

接下来，我们就重点讲讲这两种个性化召回方案。

基于内容的召回

基于内容的召回，就是我们根据文章的内容，判断这篇文章是否符合用户的喜好。具体怎么做呢？我们可以判断用户画像和文章画像中的标签或关键词是否相同，如果相同，就说明这篇文章的内容符合用户喜好，那我们就可以召回这篇文章。

这个时候，我们要解决的问题，其实就又变成了我们熟悉的标签匹配问题（第 19 讲）。因此，我们完全可以用标签和关键词作为 Key，来建立倒排索引。这样，我们就能针对用户的喜好，召回内容匹配的文章了。

当然，在上一讲中我们也说了，基于标签的召回可能会漏掉许多候选集合。因此，我们可以使用向量空间模型，将标签匹配改为高维向量空间的最近邻检索问题。这样，我们就能达到更灵活的召回目的了。

以上就是基于内容的召回的具体方法，对于向量空间的最邻近检索，我们在前面已经详细讲过，这里就不再重复了。不过，如果要将基于内容召回的技术用在推荐系统中，我们就需要充分理解它的特点和效果。那接下来，我们就来说说基于内容的召回在数据依赖、个性化和冷启动方面的优缺点。

首先，优点有 3 个，分别是**不需要其他用户数据，可以针对小众用户给出个性化的推荐，以及可以推荐冷启动的新文章。**同样的，缺点也有 3 个，分别是**依赖于用户画像系统和文章画像系统，无法挖掘出用户的潜在兴趣，以及无法给冷启动的新用户推荐文章。**这些优缺点出现的原因，我在下面的表格中都给出了解释，理解起来应该不难，那我在这里就不多说了。

	优点	缺点
数据依赖	**不需要其他用户数据：**只需要根据用户自己的用户画像和文章画像进行匹配，召回环节会更简单高效。	**依赖于用户画像系统和文章画像系统：**提前在离线阶段，分析处理用户画像和文章画像。这本身就是两个外部的复杂系统，会耗费大量的资源。
个性化	**可以针对小众用户给出个性化的推荐：**由于不需要其他用户数据，因此，不会被大众的喜好所影响，能更好地满足小众用户的需求。	**无法挖掘出用户的潜在兴趣：**尽管不使用其他用户的数据来推荐，能符合小众用户的喜好，但也容易陷入信息茧房，无法挖掘出用户的潜的兴趣。
冷启动	**可以对于冷启动的新文章进行推荐：**新上线文章因为缺乏用户的浏览、点击等行为，无法利用大众用户的喜好进行推荐。但基于内容的召回可以通过分析文章的内容，提炼出标签，然后推荐给能匹配上标签的用户。	**对于冷启动的新用户，无法推荐文章：**新注册的用户没有历史行为，我们无法给他打上标签，也就无法根据标签来给他推荐文章。针对这个问题，许多系统的解决办法是，在用户注册的时候，先让用户自己选择感兴趣的标签。

极客时间

优缺点对照表

基于协同过滤的召回

协同过滤是推荐引擎中最具有代表性的方法。协同过滤和基于内容的召回方法最大的区别就在于，它并不依赖内容本身来进行推荐，而是基于大众用户和这篇文章的互动关系来进行推荐。

其中，协同过滤还可以分为两大类：一类是传统的基于数据统计的 **Memory-based** 的协同过滤算法，也叫做**基于邻域的算法**，代表算法有**基于用户的协同过滤**（User CF，User Collaboration Filter）和**基于物品的协同过滤**（Item CF，Item Collaboration Filter）；另一类是升级版的**基于模型的 Model-based 的协同过滤算法**。今天，我们还是将重点放在协同过滤的基础算法，也就是 Memory-based 上，其他的就先不展开了。

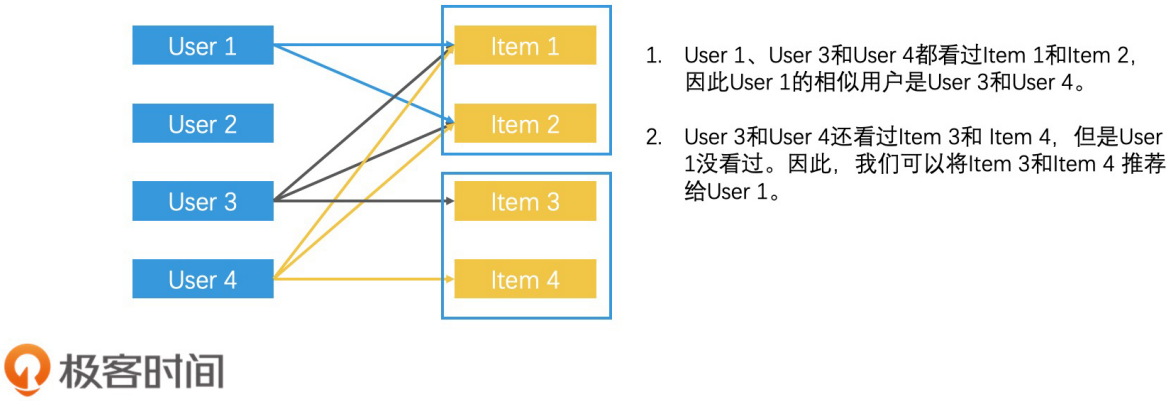
1. 基于用户的协同过滤

基于用户的协同过滤的思想其实并不复杂，说白了就是将和你相似的用户看过的文章也推荐给你。那在具体操作的时候会分为两步：

1. 找到和你最相似的一批用户
2. 将这批用户看过，但你没看过的文章推荐给你

接下来，我们通过一个例子，来直观感受一下基于用户的协同过滤过程是怎么样的。

首先，如果 User 1 看过 Item 1 和 Item 2，而 User 3 和 User 4 也看过 Item 1 和 Item 2，那么 User 1 和 User 3、User 4 就是相似用户。这样一来，如果 User 3 和 User 4 还分别看过 Item 3 和 Item 4，我们就可以将 Item 3 和 Item 4 都推荐给 User 1 了。



基于用户的协同过滤

在这个过程中，定义相似的用户并且将它们找出来是最重要的一步。那具体怎么做呢？我们先将上面的例子变成一个表格，这样看起来更清晰。

	Item 1	Item 2	Item 3	Item 4
User 1	1	2		
User 2			1	
User 3	2	1	1	
User 4	1	1		2

极客时间

你会看到，这个表格里的每篇文章下面都对应着一些数字，这其实就是每个用户对每篇文章的喜爱程度（具体可以通过用户的点击次数、收藏、评论和转发等行为计算出来的）。

那基于这张表，如果要找出和 User 1 相似的用户，我们可以将 Item 1 到 Item n 看作是一个 n 维空间，那每个用户都可以表示为 n 维空间中的一个向量，然后把他对这个物品的喜好程度，作为每个维度上的值。

这样一来，如何找到最相似的一批用户的问题，就变成了如何在 n 维向量空间中，找到和 User 1 这个点最接近的点的问题。对于向量相似度，我们一般使用余弦距离来计算。

$$\cos(X, Y) = \frac{X * Y}{||X|| * ||Y||} = \frac{x1*y1+x2*y2+x3*y3+ \ldots\ldots + xn*yn}{\sqrt{x1^2+x2^2+x3^2+ \ldots\ldots+xn^2} * \sqrt{y1^2+y2^2+y3^2+ \ldots\ldots+yn^2}}$$

极客时间

余弦距离公式

计算的过程我就不多说了，我直接把计算出来的相似度的结果告诉你。

用户	向量	和User 1的相似度
User 1	<1，2，0，0>	-
User 2	<0，0，1，0>	0
User 3	<2，1，1，0>	0.73
User 4	<1，1，0，2>	0.54

极客时间

因此，这三个用户和 User 1 的相似度的排序就是 User 3>User 4>User 2。根据这个方法，我们就能取 Top K 个相似用户，然后将他们看过的 Item 取出来进行排序推荐了。那在这个例子中，我们就可以取 User 3 和 User 4 作为相似用户，然后将 Item 3 和 Item 4 取出来排序推荐。

那么问题来了，我们是应该先推荐 Item 3 还是 Item 4 呢？这个时候，我们可以将相似用户对于做个物品的喜好进行加权打分累加，然后优先推荐分数最高的。接下来，我们就分别计算一下 Item 3 还是 Item 4 的推荐打分。

Item 3 的推荐打分是：1*0.73=0.73（User 3 的喜好度*User 3 和 User 1 的相似度）。

Item 4 的推荐打分是：2*0.54 = 1.08（User 4 的喜好度*User 4 和 User 1 的相似度）。

	Item 1	Item 2	Item 3	Item 4
User 1	1	2	1*0.73	2*0.54
User 2			1	
User 3	2	1	1	
User 4	1	1		2

极客时间

因此，根据计算得到的结果，我们会优先推荐 Item 4，再推荐 Item 3。

这就是基于用户的协同过滤的算法思想了。不过，如果要实时计算当前用户和所有其他用户的相似程度，这其实是一个遍历的操作，会非常耗时。对此，推荐系统有两种解决方案，分别是**把相似计算放在离线环节，以及在实时阶段使用向量检索来近似地完成计算更新**。下面，我们一一来看。

第一种方案，将相似计算放在离线环节。

具体来说就是，我们可以在离线环节为每个用户计算出这样一个推荐列表，然后使用 Key-value 数据库（如使用 Redis）把它加载到在线检索部分。这样，当 User 1 打开 App 时，我们通过查询这个 Key-value 数据库，就可以快速查询到推荐的文章列表了。而且，这个

推荐列表可以通过周期性的重新全量计算来完成更新。这个方案的优点是实时环节非常简单，就是一个查表操作，但是缺点是更新不够及时。

第二种方案，在实时阶段使用向量检索来近似地完成计算更新。

在第一步“寻找相似用户”的时候，我们不需要精确地计算和每个用户的相似度，而是可以使用聚类 + 倒排索引 + 乘积量化的方案，来快速地检索出和 User 1 最近邻的 k 个用户，然后将这些用户喜好的物品取出来，进行加权打分并排序即可。

你看，这不就又变成了我们熟悉的实时“召回 + 打分排序”的过程了嘛。这个方案的优点是实时性很好，只要用户有了变化，就能马上反馈出来，但是缺点是实时阶段的检索过程很复杂，并且由于采用了非精准的近邻检索技术，因此结果也不够精确。

总结来说，基于用户的协同过滤既不依赖于文章本身的属性挖掘，也不会造成用户的兴趣被局限在历史的兴趣范围中。它能将用户没看过，但是其他相似用户喜欢的文章推荐出来，因此特别适用于资讯类的平台。不过它也有自己的不足，它适用于用户数不太大的场合。因为一旦用户数太大，计算所有的用户两两之间的相似度，也会是一个非常巨大的开销。

2. 基于物品的协同过滤

基于物品的协同过滤，简单来说就是基于你之前看过的物品，找出相似的物品并进行推荐。它的实现也分为两步：

1. 计算出用户看过的每个物品的相似物品
2. 将这些相似物品进行排序，然后再推荐

我们还是通过上面的例子，来进一步分析这个过程。假设，User 1 看过 Item 1 和 Item 2，那我们就需要寻找 Item 1 的相似物品和 Item 2 的相似的物品，将它们推荐给 User 1。

所以，这次我们要将刚才的表格换一个维度来看，也就是把 User 和 Item 的位置对调，重新画出这个表格。

	User 1	User 2	User 3	User 4
Item 1	1		2	1
Item 2	2		1	1
Item 3		1	1	
Item 4				2

极客时间

由于这一次是寻找相似的物品，因此，我们把 User 1 到 User n 作为维度，构建一个 n 维的向量空间，然后把每一个物品用这个向量空间中的一个向量来表示。

那么，要查找和 Item 1 相似的物品，我们就要先找出最接近的 k1 个 Item 向量，然后用 Item 1 对 User1 的权重乘上每个 Item 和 Item 1 的相似度，就能得到这 k1 个 Item 的推荐度。

同理对于 Item 2，我们也可以找出 k2 个 Item，并计算出这 k2 个 Item 的推荐度。最后，我们将 k1 个 Item 和 k2 个 Item 进行合并，将相同 Item 的推荐度累加，就能得到每个 Item 对于 User 1 的推荐度了。具体的计算过程我就不详细列出来了，你可以自己算一下。

这就是基于物品的协同过滤的算法思想了。在实际实现的时候，推荐引擎为了加快检索效率，会将 2 个步骤分别放在两个环节。

首先，将第一步“寻找每个物品的相似物品列表”放在离线环节。具体的操作是以 Item ID 为 Key，以相似物品列表为 posting list，来生成倒排索引，再把它存入线上的 Key-value 数据库中。

其次，是把“对相似物品进行排序”放在实时环节。这样一来，当我们需要对一个用户推荐新的物品时，只以这个用户看过的 Item 为 Key，去 Key-value 数据库中取出所有推荐的 Item 列表，然后将这些 Item 合并后排序即可。

你会发现，通过这样的设计，基于物品的协同过滤算法就能简单地支持实时反馈了。那总结来说，和基于用户的协同过滤算法相比，基于物品的协同过滤算法更注重于用户的兴趣传

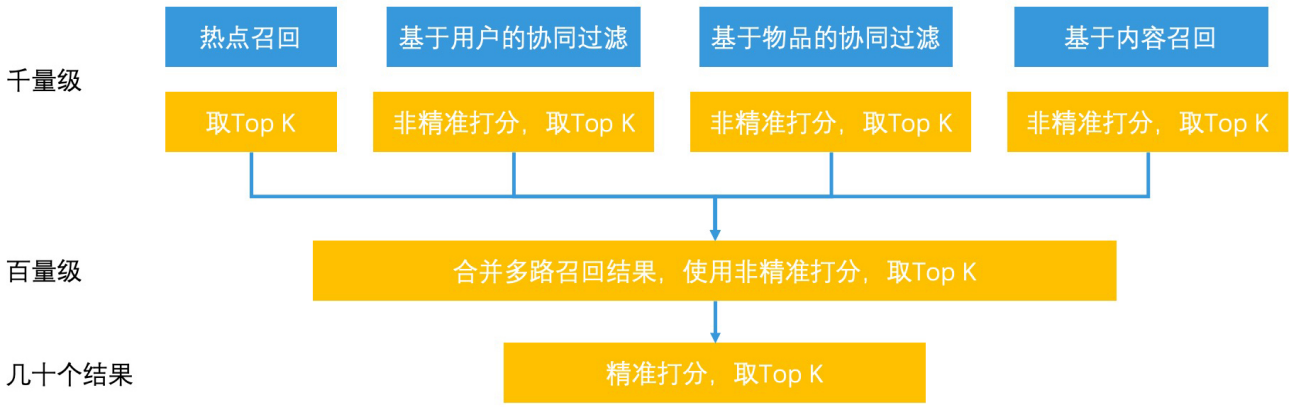
承，而基于用户的协同过滤算法会更注重于社会化的推荐。因此，新闻资讯类的 App 更倾向于使用基于用户的协同过滤算法，而电商平台更倾向于使用基于物品的协同过滤算法。

如何对多种召回方案进行选择和排序？

通过前面的分析，你会发现，不同的推荐算法根据各自的特点，会召回不同的候选文章。那我们应该选择哪种方案呢？

实际上，因为推荐引擎并没有检索限定条件，所以它可以从不同的维度来进行推荐，而且不同的用户对于不同的推荐方案也有不同的接受度。综合来说，我们很难说哪一种方案一定就是最好的。因此，在推荐引擎的实现中，我们更多的是采用**混合推荐法，也就是一并使用上面的所有方案**。

那这就带来了一个问题：每种召回方案都会返回大量的候选集，这会使得系统难以承受排序计算的代价。为了解决这个问题，推荐引擎中采用了分层打分过滤的排序方式。



极客时间

分层打分过滤示意图

下面，我就结合上面给出的示意图来说一下分层打分过滤的过程。

首先，每一个召回通路都会使用自己的非精准打分算法，截取千级别之内的候选集。然后，推荐引擎会合并这多个召回通路截取的几千个结果，也就是使用简单的机器学习模型进行非精准打分，选出最好的上百个结果。最后，推荐引擎会使用精准的深度学习模型，选出最好的几十个结果返回给用户。这就是用户看到的最终的推荐结果了。

以上，就是推荐引擎从召回到排序的完整检索过程了。

重点回顾

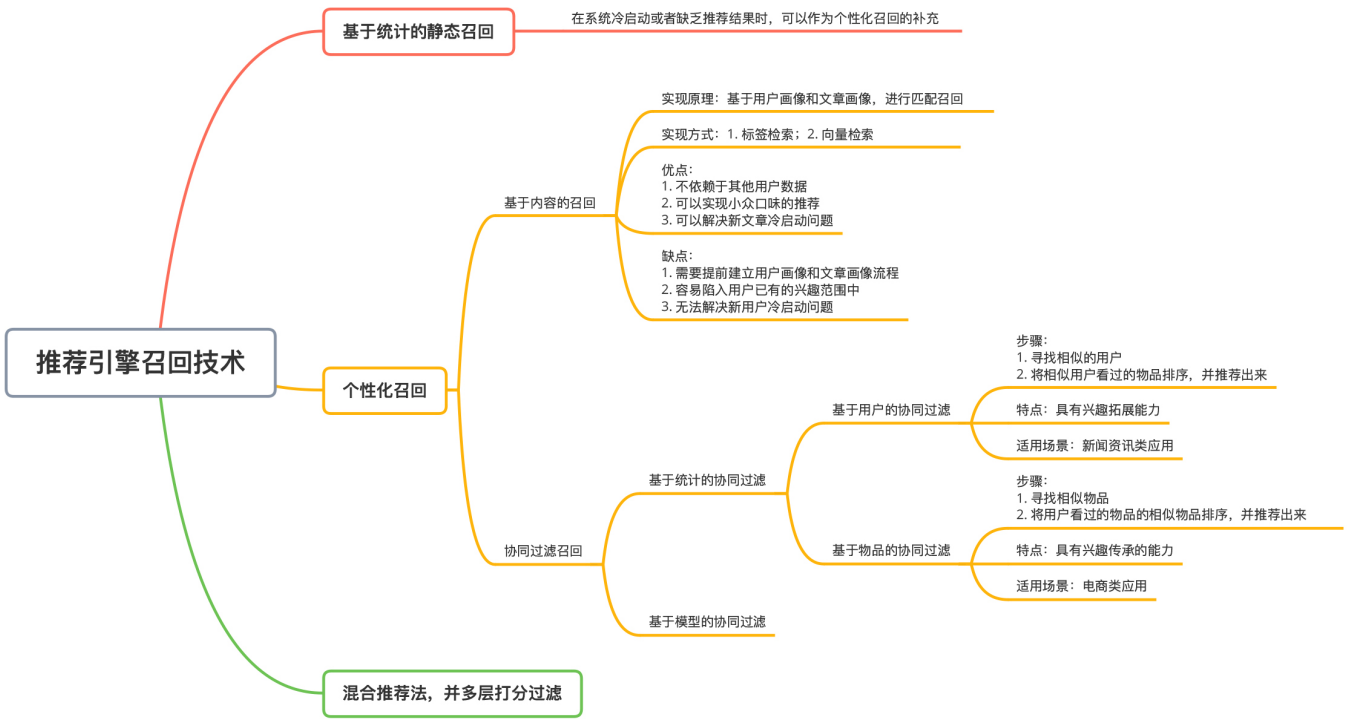
今天，我们重点学习了推荐引擎中的个性化召回算法，它又为基于内容的召回和基于协同过滤的召回。我们来总结一下它们各自的特点。

首先，基于内容的召回，本质上是基于用户画像和文章画像进行匹配召回。在实际操作中，我们可以使用标签检索和向量检索来完成基于内容的召回。

然后，基于协同过滤的召回我们重点讲了基于用户的协同过滤和基于物品的协同过滤。基于用户的协同过滤，会先寻找和当前用户最相似的用户，再将这些用户看过的物品推荐出来；而基于物品的协同过滤，则是先整理出相似的物品列表，再根据当前用户看过的物品，找出对应的物品列表，最后进行合并推荐。

总的来说，这两种协同过滤算法都是先寻找最邻近的“邻居”，再进行打分排序。也就是说，**这两种协同过滤算法抽象起来看，依然是使用“检索 - 排序”的检索技术来实现的。**

此外，在实际工作中，推荐引擎会同时使用多种召回技术进行混合推荐。而在使用混合推荐法的时候，系统会进行多层的打分过滤，来保证检索性能。



课堂讨论

1. 对于文章中提到的基于物品的协同过滤召回，以文中的数据为例子，你能按文中介绍的方式，使用余弦距离计算出每个 Item 和 Item 1 的相似度吗？
2. 关于搜索引擎、广告引擎、推荐引擎，你觉得它们有哪些设计可以相互借鉴？为什么？

欢迎在留言区畅所欲言，说出你的思考过程和最终答案。如果有收获，也欢迎把这一讲分享给你的朋友。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　19 | 广告系统：广告引擎如何做到在0.1s内返回广告信息？

下一篇　结束语 | 成长和进化，技术如此，我们亦如此

精选留言 (3)

写留言

一步

2020-05-17

将第一步“寻找每个物品的相似物品列表”放在离线环节。具体的操作是以 Item ID 为 Key，以相似物品列表为 posting list，来生成倒排索引，再把它存入线上的 Key-value 数据库中

利用这种方法的时候，每个item 都会生成一个 posting list, 这样的每个商品都会保存 n...

展开

作者回复: 如果每个posting list都存了所有的商品，的确会比较浪费空间，而且其实也没必要，因为最后只会选取top k个。因此，我们可以对posting list进行截断，仅保留相似度超过一定阈值的item就好了。这样就能大幅减少空间。

一步

2020-05-17

我们就要先找出最接近的 k1 个 Item 向量，然后用 Item 1 对 User1 的权重乘上每个 Item 和 Item 1 的相似度，

这个应该是： 每个 Item 对 User1 的权重乘上每个 Item 和 Item 1 的相似度

展开

作者回复: 并不是哦。在基于物品的协同过滤中，对于要推荐的item，我们之前并不知道这个item和user1有什么关系，我们知道的是两个信息:
一个是user1对item1的打分(记为w)，另一个是item1和item的相似度(记为s)，因此，user1和item的相关性，就是w*s

范闲

2020-05-15

1.关于第一个问题，其实有点疑问。每个item的向量实际上是不同用户偏好构成的。最终对不同item求出来实际上对于某个item，用户偏好相近的程度

2.整体技术上的架构是类似的，但是在场景上区分很大。

展开

作者回复: 1.第一个问题，的确，最后针对某个用户推荐的时候，我们是需要具体计算和这个指定用户的相关性的。不过我在文中也说过，item based方案的实现会分为两部分，第一部分是离线构建起相似item的倒排表，第二部分才是在线上环节，根据具体用户，找出推荐item。我在这道

题中，其实只要求做第一步，给出每个item对应的posting list即可。如果你想把第二步对具体用户推荐也完成的话，可以以用户1为例子，看看会推荐出什么item。